SOLUTIONS FOR IMPLEMENTING THE N-BODY SIMULATION ON THE PASCAL COMPUTE UNIFIED DEVICE ARCHITECTURE

Dana-Mihaela Petroşanu [1*]
Alexandru Pîrjan [2]

ABSTRACT

In this paper, we develop and propose novel solutions for implementing the N-body simulation on the latest Pascal parallel processing Compute Unified Device Architecture (CUDA) as to attain a high level of performance and efficiency. The innovative aspect of our research emerges from the development and implementation of the N-body simulation on the latest Pascal Compute Unified Device Architecture, making use of the latest features of the CUDA Toolkit 8.0, employing the architecture's dynamic parallelism feature in order to effectively manage the unbalancing of the processing tasks that appears once the number of corresponding bodies differs throughout the processing threads.

KEYWORDS: *Graphics Processing Unit (GPU), N-Body Simulation, Verlet-Leapfrog Algorithm, CUDA, Dynamic Parallelism.*

1. INTRODUCTION

Within this paper, we have proposed and developed innovative solutions that facilitate the implementation of the N-body simulation ($N \geq 2$) on the latest Pascal Compute Unified Device Architecture (CUDA), launched in 2016 by the NVidia company. Of particular interest when developing the implementation was to attain a high level of performance and efficiency. An efficient implementation of the N-body simulation has multiple applications ranging from astrophysical simulation to a variety of computational tasks in numerous scientific fields such as: fluid mechanics, medicine and computer graphics applications.

In [1], the authors develop an implementation of the N-body simulation on the GeForce GTX8800 NVidia Graphics Processing Unit. In [2], the authors depict how GPUs can be used for N-body simulations, in order to obtain improvements in performance over the Central Processing Units available in 2006. Thus, on an ATI X1900XTX, they develop an algorithm for performing the force computations that represent the most part of stellar and molecular dynamics simulations. In [3], the authors develop an implementation of N-body simulation on the Intel Knights Landing Central Processing Unit architecture.

[1*] corresponding author, PhD Lecturer Department of Mathematics-Informatics, University Politehnica of Bucharest, 313, Splaiul Independentei, district 6, code 060042, Bucharest, Romania, danap@mathem.pub.ro
[2] PhD Hab, Associate Professor Faculty of Computer Science for Business Management, Romanian-American University, 1B, Expozitiei Blvd., district 1, code 012101, Bucharest, Romania, alex@pirjan.com

Even though more implementations of the N-body simulation exist in the scientific literature, most of these are confronted with serious limitations originating from the required huge computational processing power. The novelty of our approach resides in developing and implementing the N-body simulation on the latest parallel processing Pascal Compute Unified Device Architecture, benefitting from the most powerful features of the CUDA Toolkit 8.0, like the dynamic parallelism feature that helps us to solve efficiently the unbalancing of the processing tasks that appears once the number of corresponding bodies differs throughout the processing threads.

We have used the dynamic parallelism feature to call an additional kernel function with the purpose of processing in parallel the last states of the bodies, consequently attaining a high level of performance and efficiency for the developed solution. Although there are many works in the literature that implement the N-body simulation, to our best knowledge, up to this moment none of them have implemented the N-body simulation on the latest Pascal Compute Unified Device Architecture, making use of the architecture's dynamic parallelism feature.

2. THE N-BODY SIMULATION IN THE ALL-PAIRS APPROACH

In the following we depict the all-pairs approach of the N-body simulation ($N \geq 2$), a technique based on the evaluation of all the interactions, considering all the possible pairs for each of the N considered bodies [4]. Thus, the number of interactions is $\frac{N(N-1)}{2}$. In the following, we denote the vectors with lowercase bold letters. Thus, for each positive integer $i, 1 \leq i \leq N$, we denote by $\boldsymbol{x_i}$ the initial 3D position and by $\boldsymbol{v_i}$ the initial velocity for each of the N bodies, by $\boldsymbol{f_{ij}}$ the force vector caused on the i-body by the gravitational attraction of the j-body.

According to the Newton's law of universal gravitation, each of the N bodies attracts every other body with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between them [5], as depicted in the following equation:

$$f_{ij} = G\frac{m_i m_j}{{r_{ij}}^2}, \forall\, 1 \leq i, j \leq N \tag{1}$$

where the indexes i, j refer to two bodies, having the mases m_i and m_j, r_{ij} the module of the vector that has the origin at the body i and the extremity at the body j, f_{ij} the module of the force vector (the magnitude of the force), while G is the gravitational constant $6.67408(31)\times10^{-11} m^3 \cdot kg^{-1} \cdot s^{-2}$. This equation can be written in the vector form, that highlights the directions of the $\boldsymbol{f_{ij}}$ and $\boldsymbol{r_{ij}} = \boldsymbol{x_j} - \boldsymbol{x_i}$ vectors, using the unit vector $\frac{\boldsymbol{r_{ij}}}{r_{ij}}$:

$$\boldsymbol{f_{ij}} = G\frac{m_i m_j}{{r_{ij}}^2}\frac{\boldsymbol{r_{ij}}}{r_{ij}}, \forall\, 1 \leq i, j \leq N \tag{2}$$

Taking into account all the possible pairs between the body i ($1 \leq i \leq N$) and all the others N-1 bodies, one obtains the total force $\boldsymbol{F_i}$ as the sum of all the interactions, represented by the $\boldsymbol{f_{ij}}$ functions (except the case when $i = j$, because in this case the denominator of the equation (2) becomes zero):

$$\boldsymbol{F_i} = \sum_{\substack{1\le j\le N \\ i\neq j}} \boldsymbol{f_{ij}} = Gm_i \sum_{\substack{1\le j\le N \\ i\neq j}} \frac{m_j \boldsymbol{r_{ij}}}{{r_{ij}}^3} \quad (3)$$

Under the effect of the interactions, the bodies tend to move from their initial positions, approaching each other. As the r_{ij} distances decrease, the $\boldsymbol{F_i}$ forces grow without any limit. This growing could become an impediment when applying numerical methods for integration [6]. Generally, when using the N-body simulations in astrophysics, the collisions between the N considered bodies are not possible. Even if the r_{ij} distances decrease and tend to zero, the bodies (that represent galaxies), do not collide but pass near or through each other. In order to avoid the unlimited growing of the $\boldsymbol{F_i}$ forces, one adds a softening positive factor, denoted by ϵ^2, at the denominator of the equation (3):

$$\boldsymbol{F_i} \approx Gm_i \sum_{1\le j\le N} \frac{m_j \boldsymbol{r_{ij}}}{({r_{ij}}^2+\epsilon^2)^{3/2}} \quad (4)$$

When adding this factor, the condition $i \neq j$ is no longer necessary, because when $i = j$, the vector $\boldsymbol{r_{ii}} = 0$ and therefore, the force $\boldsymbol{f_{ii}} = G\frac{m_i m_i \boldsymbol{r_{ii}}}{({r_{ii}}^2+\epsilon^2)^{3/2}}$ becomes zero, while the denominator of $\boldsymbol{f_{ii}}$ is not zero. Using the softening factor, the magnitude of the interactions between the bodies are limited and thus, the numerical integration is facilitated.

Using the equation (4), one can express the acceleration of the i-body as:

$$\boldsymbol{a_i} = \frac{\boldsymbol{F_i}}{m_i} \approx G \sum_{1\le j\le N} \frac{m_j \boldsymbol{r_{ij}}}{({r_{ij}}^2+\epsilon^2)^{3/2}} \quad (5)$$

Taking into account the nature of the problem that is modeled using the N-body simulation ($N \geq 2$), one can choose different integration methods [7]. In our case, in order to obtain the current positions and velocities of each body, we have used the Verlet-Leapfrog Algorithm, a computationally efficient algorithm applicable to our problem. This algorithm is frequently used in molecular dynamics simulations, in N-body simulations problems, in computer graphics.

The Verlet-Leapfrog Algorithm consists in a numerical method, useful for integrating the Newton's equations of motion. Even if this method was previously used in 1792 by the French mathematician and astronomer Jean Baptiste Joseph Delambre, it is consecrated as the Verlet's algorithm, who has used it in molecular dynamics in 1967. This approach was also applied in 1909 by Cowell and Crommelin, in order to compute the orbit of Halley's Comet and in 1907 by Carl Störmer, in his study regarding the electrical particles' trajectories in a magnetic field.

The Verlet integration method has some important advantages, arising from its properties [8]. Thus, it provides a good numerical stability and time reversibility. Considering the unpredictable nature of individual atoms or molecules motion, an important problem that arises when modeling problems related to this motion is to use accurate and stable integration schemes for the obtained ordinary differential equations. Moreover, the number of equations can be very large, as they are 6 for each particle (3 for the components of the position vector and 3 for the components of the velocity).

If considering N particles, the total number of equations is $6N$, while the number of interaction terms is $\frac{N(N-1)}{2}$ in the case of pair-wise interactions. As a consequence, one must use an algorithm that reduces to the minimum the necessary number of evaluations that must be made on the right side of the obtained ordinary differential equations.

The Verlet integration schemes satisfy all of the above-mentioned requirements and comprise three main different algorithms: Basic, Leapfrog and Velocity. After analyzing and testing them, we have chosen for solving our problem the Verlet Leapfrog Algorithm as it has offered the best results, features and implementation opportunities. In the following, we will describe this algorithm.

The Verlet Leapfrog Algorithm can be obtained using a Taylor expansion of the position $r(t)$ in t, to order Δt^2, where Δt is the time step:

$$r(t+\Delta t) = r(t) + r'(t)\Delta t + \frac{1}{2}r''(t)\Delta t^2 + \mathcal{O}(\Delta t^3) \quad (6)$$

$$r(t-\Delta t) = r(t) - r'(t)\Delta t + \frac{1}{2}r''(t)\Delta t^2 - \mathcal{O}(\Delta t^3) \quad (7)$$

By adding and subtracting the equations (6) and (7), one obtains:

$$r(t+\Delta t) + r(t-\Delta t) = 2r(t) + r''(t)\Delta t^2 + \mathcal{O}(\Delta t^3) \quad (8)$$

$$r(t+\Delta t) - r(t-\Delta t) = 2r'(t)\Delta t + \mathcal{O}(\Delta t^3) \quad (9)$$

In the following, we define:

$$\partial r(t) = r(t+\Delta t) - r(t) \quad (10)$$

Taking into account the equation (6), the equation (10) can be written:

$$\partial r(t) = r'(t)\Delta t + \frac{1}{2}r''(t)\Delta t^2 + \mathcal{O}(\Delta t^3) \quad (11)$$

The equation (10) implies that:

$$\partial \mathrm{r}(\mathrm{t}-\Delta \mathrm{t}) = \mathrm{r}(\mathrm{t}) - \mathrm{r}(\mathrm{t}-\Delta \mathrm{t}) \quad (12)$$

Taking into account the equation (7), the equation (12) can be written as:

$$\partial \mathrm{r}(\mathrm{t}-\Delta \mathrm{t}) = \mathrm{r}'(\mathrm{t})\Delta \mathrm{t} - \frac{1}{2}\mathrm{r}''(\mathrm{t})\Delta \mathrm{t}^2 + \mathcal{O}(\Delta \mathrm{t}^3) \quad (13)$$

By subtracting the equations (11) and (13), one obtains:

$$\partial r(t) - \partial r(t-\Delta t) = r''(t)\Delta t^2 + \mathcal{O}(\Delta t^3) \quad (14)$$

Equation (14) can be written as:

$$\partial r(t) = \partial r(t-\Delta t) + r''(t)\Delta t^2 + \mathcal{O}(\Delta t^3) \quad (15)$$

Afterwards, by adding the equations (11) and (13), one obtains:

$$\partial r(t) + \partial r(t-\Delta t) = 2\, r'(t)\Delta t + \mathcal{O}(\Delta t^3) \quad (16)$$

From the equation (16) one obtains:

$$r'(t) = \frac{\partial r(t) + \partial r(t-\Delta t)}{2\,\Delta t} + \mathcal{O}(\Delta t^2) \quad (17)$$

Taking into account the equation (10), the equation (17) can be written as:

$$r'(t) = \frac{\partial r(t-\Delta t)+r''(t)\Delta t^2+\partial r(t-\Delta t)+\mathcal{O}(\Delta t^3)}{2\,\Delta t} + \mathcal{O}(\Delta t^2) \tag{18}$$

or, after calculations,

$$r'(t) = \frac{\partial r(t-\Delta t)}{\Delta t} + \frac{1}{2}\, r''(t)\Delta t + \mathcal{O}(\Delta t^2) \tag{19}$$

In the following, the Verlet Leapfrog Algorithm uses the half time step in order to obtain accurate velocities. Thus, one defines:

$$v_{n+\frac{1}{2}} = v\left(t + \frac{1}{2}\Delta t\right) \tag{20}$$

where $v(t) = r'(t)$ is the velocity. Using a Taylor expansion of the right member of the equation (20), one obtains:

$$v_{n+\frac{1}{2}} = v(t) + \frac{1}{2}v'(t)\Delta t + \mathcal{O}(\Delta t^2) \tag{21}$$

Similarly, considering:

$$v_{n-\frac{1}{2}} = v\left(t - \frac{1}{2}\Delta t\right) \tag{22}$$

and using a Taylor expansion of the right member of the equation (22), one obtains:

$$v_{n-\frac{1}{2}} = v(t) - \frac{1}{2}v'(t)\Delta t + \mathcal{O}(\Delta t^2) \tag{23}$$

By dividing equation (11) with Δt one obtains:

$$\frac{\partial r(t)}{\Delta t} = r'(t) + \frac{1}{2}r''(t)\Delta t + \mathcal{O}(\Delta t^2) \tag{24}$$

Taking into account the definition of $v(t) = r'(t)$ and comparing the equations (21) and (24) one can conclude that:

$$v_{n+\frac{1}{2}} = v\left(t + \frac{1}{2}\Delta t\right) = \frac{\partial r(t)}{\Delta t} \tag{25}$$

By dividing equation (13) with Δt one obtains:

$$\frac{\partial r(t-\Delta t)}{\Delta t} = r'(t) - \frac{1}{2}r''(t)\Delta t + \mathcal{O}(\Delta t^2) \tag{26}$$

Considering the definition of $v(t) = r'(t)$ and comparing the equations (23) and (26) one can conclude that:

$$v_{n-\frac{1}{2}} = v\left(t - \frac{1}{2}\Delta t\right) = \frac{\partial r(t-\Delta t)}{\Delta t} \tag{27}$$

In conclusion, the velocities at time t can be computed by adding equations (21) and (23):

$$v_n = \frac{v_{n+\frac{1}{2}}+v_{n-\frac{1}{2}}}{2} + \mathcal{O}(\Delta t^2) \tag{28}$$

Subtracting the equations (21) and (23), one obtains:

$$v_{n+\frac{1}{2}} - v_{n-\frac{1}{2}} = v'(t)\Delta t + \mathcal{O}(\Delta t^2) \quad (29)$$

Using equation (25) one obtains:

$$\partial r(t) = r(t + \Delta t) - r(t) = v_{n+\frac{1}{2}}\,\Delta t \quad (30)$$

Using the notations $r_n = r(t)$, $r_{n+1} = r(t + \Delta t)$, the equation (30) can be written as:

$$r_{n+1} = r_n + v_{n+\frac{1}{2}}\,\Delta t \quad (31)$$

The equations (28) and (29) give the velocities, while the equation (31) gives the positions of the particles. In conclusion, in the leapfrog method the position is calculated at time intervals that have the dimension of an integer multiple of the time step, $t, t + \Delta t, t + 2\Delta t$, ..., while the velocity is evaluated at the times $t, t + \frac{1}{2}\Delta t, t + \frac{3}{2}\Delta t$, ..., between these points, starting from an initial point t. The above-mentioned leapfrog algorithm, requires less storage and is less expensive than other approaches when judging it from the computational requirements point of view [8]. In the case of large scale computations, these aspects represent important advantages for the programmer. The Verlet Leapfrog Algorithm has also the advantage that, even at large time steps, the conservation of energy is respected. Therefore, when this algorithm is used, one can considerably decrease the computation time.

In the following, we depict our implementation of the N-body simulation ($N \geq 2$) on the Pascal Compute Unified Device Architecture.

3. IMPORTANT ASPECTS REGARDING THE DEVELOPMENT OF THE N-BODY SIMULATION IN THE PASCAL COMPUTE UNIFIED ARCHITECTURE

The most important aspects that we had to take into account when developing the N-body simulation in the CUDA architecture comprised the proper management of the synchronization process, of the atomic operations, of the race situations, as to circumvent memory leakage and achieve a sufficient amount of dynamic parallelism. When one develops N-body simulations ($N \geq 2$) that are targeted by the central processing unit (CPU) and for which a single processing thread is sufficient, the whole process of managing race conditions is extensively simplified. In these situations, the developer must examine carefully the data flow as to identify if a specific value has been loaded from a certain variable before storing the latest updated value in it.

The vast majority of the compilers that are being used these days for compiling software applications that make use of a single processing thread have the technical capability to pinpoint precisely these issues. When developers are programming applications that use multiple processing execution threads, race conditions have to be methodically and accurately averted. The threading mechanism implemented in CUDA is configured as to achieve a high degree of performance without taking into consideration a chronological order in which the kernels have been invoked and the threads executed.

Like in the case of the N-body simulation, when the state of an element at a certain step is influenced by the computed result from a previous step, if the developer allocates a processing thread for each body, for the result to be correct the threads would have to be

processed in an ascending order and the results of the previous execution steps to have already been calculated and stored in the corresponding variables. When more execution threads are processed in parallel, the risks become higher for the outcome to have errors, in some situations the whole application might even crash.

What makes it more complicated is the fact that in some random situations the program might produce a correct output if a processing thread has the possibility to compute and store the result before another thread needs to retrieve it. All of these specific aspects provide a valuable insight on the issue of a race condition, when certain functions of an application are processing data simultaneously to a particular point in the execution path. Therefore, we had to develop the N-body CUDA implementation by employing a synchronization process as the order of execution in the device can vary to a great extent.

By using the synchronization process, we were able to transfer data between the threads belonging to the same block and between multiple blocks that were part of the same grid of thread blocks. We have used the local memory area and register memory available to each of the threads. The shared memory that exists at the level of a block of threads helped us to interchange data between the threads that resided in the same block.

When developing the N-body simulation ($N \geq 2$), we have used the "cuda-memcheck" software instrument with the aim of identifying, isolating and solving the issues concerning the memory leaks and over-usage of memory. In developing the N-body implementation, we have taken into consideration the fact that the Pascal CUDA architecture offers support for the dynamic parallelism feature and thus, we were able to call, from an initial CUDA kernel, supplementary child CUDA kernels and synchronize the processing. This technique helped us avoid having to invoke more kernel functions or to keep always several threads idle for being used in the final steps of the computation. By using the dynamic parallelism technique, we were able to save a huge amount of computational resources and avoid inefficient results when computing a large number of bodies.

Of particular usefulness when implementing the simulation using the dynamic parallelism solution was the fact that we were able to configure and execute grids of blocks, containing more processing threads and in the same time to postpone further processing, until all the grids of the blocks have finalized the processing up to the precision of a processing thread residing within a block of the grid. Therefore, we were able to program a thread from within a grid of blocks, so that in certain situations to have the possibility of invoking a new grid of thread blocks (a child grid of thread blocks) that belongs to the initial parent grid of thread blocks. An advantage of the dynamic parallelism solution that we have implemented in our approach consists in the nesting mechanism that is implemented in the architecture and automatically checks that a parent grid completes the processing only after the child grids have completed their tasks. Therefore, we made use of the implicit synchronization mechanism that is enforced by the CUDA runtime on the parent and child kernel functions.

Through the dynamic parallelism solution that we have implemented, we made sure that the graphics processing units' resources were efficiently spent and that an appropriate occupancy of the available resources was achieved, as the child kernel functions that were called by the parent ones were processing the tasks in parallel with minimum control

divergence or even none whatsoever. When the number of bodies is small, we have programmed the solution to process using only the parent kernel and not to invoke a supplementary child kernel as there is not sufficient parallelism in this case to warrant the invoking of other functions.

When implementing the N-body simulation ($N \geq 2$) in the Pascal CUDA architecture, after having divided the tasks, we have allocated them to more blocks of processing threads. We have tested extensively different methods for allocating the sizes of the grids and of the processing blocks and we have reached peak performance using the following approach:

$$\mathrm{NATB} = \begin{cases} \left[\frac{\mathrm{N}}{512}\right] + 1, \text{if N does not divide with } 512 \\ \frac{\mathrm{N}}{512}, \text{if N divides with } 512 \end{cases} \tag{32}$$

and

$$\mathrm{NTPB} = \begin{cases} \mathrm{N}, \text{if NATB} = 1 \\ 512, \text{if NATB} \geq 2 \end{cases} \tag{33}$$

where $NATB$ represents the number of allocated thread blocks, N is the number of bodies, $NTPB$ represents the number of threads per block and $[\frac{N}{512}]$ is the integral part of the real number $\frac{N}{512}$.

In the following section, we present the experimental results that we have conducted and an analysis of the obtained performance.

4. EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS OF THE N-BODY SIMULATION IMPLEMENTATION ON THE PASCAL ARCHITECTURE

We have developed and run an experimental test suite in order to check the performance of the developed implementation of the N-body simulation and we have compared the results obtained when benchmarking our developed implementation on the Pascal architecture with those provided by a state of art sequential classical implementation of the N-body simulation on the central processing unit.

In order to analyse the level of performance, we have developed and conducted a benchmark suite, using as a hardware configuration the central processing unit Intel i7-5960x operating at 3.0 GHz with 32GB (4x8GB) of 2144 MHz, DDR4 quad channel and the GeForce GTX 1080 NVIDIA graphics card with 8GB GDDR5X 256-bit from the Pascal architecture. The software configuration that we have used is Windows 10 Educational operating system and the CUDA Toolkit 8.0 with the NVIDIA developer driver.

In our experimental tests, we have successively benchmarked different cases regarding the number N of interacting bodies, ranging from 16 to 8,192 and different number of execution steps, denoted by n $(5,10,15)$. In order to ensure the accuracy of the results, we have run 100 iterations for each of the benchmark tests and afterwards we have computed the average of these results. Thus, for each of the analyzed cases, we have computed the average total execution time (measured in milliseconds) for the CPU implementation

(CPUT), for the GPU implementation (GPUT) and then we have also computed a relevant metric, the CPUT/GPUT ratio for the corresponding number of execution steps. The measured total execution times comprise the necessary time for computing, for each step, the new positions, velocities and accelerations of the N interacting bodies.

The N-body problem represents an initial value problem, comprising the system of differential equations (mentioned in section 2) and initial conditions. As our implementation is based on parallel computations, in order to obtain, in each of the analyzed situations, a comparable amount of computations, we have decided to use a specific generator for the initial conditions (3 components of the position vector, 3 components of the velocity vector, the value of the mass for each body and the softening positive factor ϵ^2). Thus, in order to randomly generate the initial conditions, we have used the newest version of NEMO (A Stellar Dynamics Toolbox, Version 3.3.2, released in March 14, 2014) [9].

As the results obtained using different settings for the number of execution steps ($n = 5, n = 10, n = 15$) have provided similar performance results, in the following we present and analyze the case when $n = 15$. We have synthetized these results, highlighting for each of the considered values of N, the number of interactions, $\frac{N(N-1)}{2}$, the total execution times registered when running the N-body simulation on the CPU (CPUT), on the GPU (GPUT) (both measured in milliseconds) and also the dimensionless CPUT/GPUT ratio (**Table 1**).

Table 1. Experimental results for $n = 15$ execution steps

No	**Number *N* of bodies**	**Number *n* of interactions**	**CPUT (ms)**	**GPUT (ms)**	**CPUT/GPUT**
1	16	120	0.29	3.731	0.07773
2	32	496	1.128	7.114	0.15857
3	64	2,016	4.472	13.666	0.32725
4	128	8,128	17.826	26.414	0.67488
5	256	32,640	70.907	52.905	1.34028
6	512	130,816	270.602	103.978	2.60249
7	1,024	523,776	874.443	208.918	4.18558
8	2,048	2,096,128	3433.961	422.939	8.11928
9	4,096	8,386,560	14013.55	847.882	16.5277
10	8,192	33,550,336	54338.387	1693.389	32.0885

In order to facilitate the comparison of the obtained experimental results, we have also represented them in **Figure 1** and **Figure 2**. Thus, in **Figure 1** we have represented the total execution times, while in **Figure 2** we have represented the CPUT/GPUT ratio, for $n = 15$ execution steps.

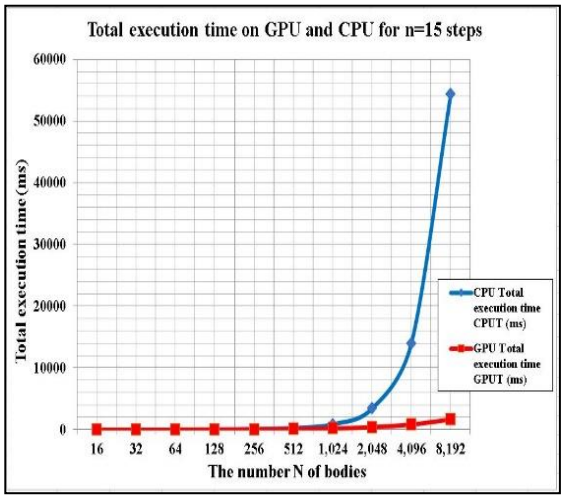

Figure 1. The total execution times for $n = 15$ execution steps

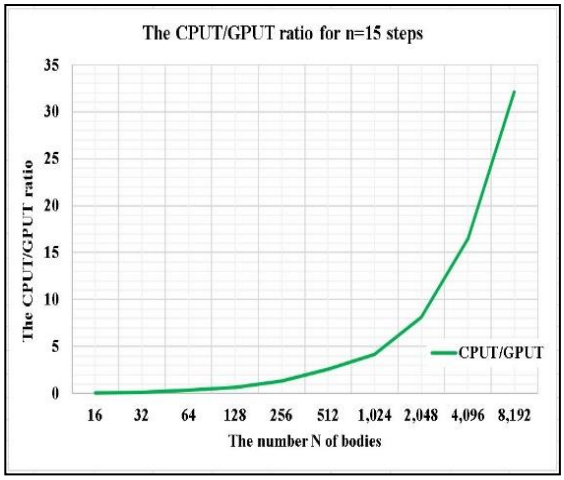

Figure 2. The CPUT/GPUT ratio for $n = 15$ execution steps

Analysing the experimental results presented in the above table and figures along with the results obtained for the other values of n, we have concluded that when the number N of bodies is less than 256, the best results (the lowest execution time) have been recorded on the CPU because in this case, the required computational load does not fully employ the parallel processing power of the GPU.

In all of the analysed situations, when the number N of interacting bodies is higher than 256, the GPU performance surpasses the CPU's one. As **Figure 2** highlights, the CPUT/GPUT ratio is sub-unitary for $N < 256$ and supra-unitary for $N \geq 256$. As the

number N of bodies increases, this ratio also increases, and its greatest value is reached in the case of $N = 8{,}192$ when the average total execution time registered by the GPU is more than 32 times lower (32.0885) than the average total execution time registered by the CPU. We have registered similar results when choosing $n = 5$ and $n = 10$ execution steps. Thus, for $n = 5$ the highest value of the dimensionless CPUT/GPUT ratio was 32.1972, while for $n = 10$ the highest value of this ratio was 32.3525, both of these values being registered for $N = 8{,}192$.

The experimental results that we have obtained outline that our solutions for implementing the N-body simulation on the latest Pascal architecture attain a high level of performance highlighted by the reduced execution times, when compared to the state of art sequential classical approach. Thus, our implementation has the ability of becoming a useful, powerful tool in a wide range of scientific domains that employ fast, accurate N-body simulations.

5. CONCLUSIONS

In our research, we have developed and proposed novel solutions for implementing the N-body simulation, harnessing the enormous parallel processing power of the latest Pascal Compute Unified Device Architecture, in order to achieve a high level of performance and efficiency. An important aspect of our research consists in employing the latest technical characteristics of the CUDA Toolkit 8.0, leveraging the architecture's dynamic parallelism feature for balancing the computational tasks.

The obtained results reflect the efficiency of the developed solutions and their suitability for implementing the CUDA Pascal N-body simulation, based on the Verlet Leapfrog Algorithm, in various scientific fields, highlighting the undisputable advantages of our solution, compared to the classical sequential approaches, when having to process a large number of interacting bodies. We have conducted extensive experimental tests, choosing various settings regarding the number N of bodies, the number n of execution steps, computing in each of the cases the average of 100 iterations, in order to obtain relevant, accurate, reliable results and a detailed analysis of our implementation. The experimental suite highlights the reliability, efficiency and applicability of our developed solution regarding the implementation of the N-body simulation in the Pascal architecture.

In the scientific literature one can find more implementations of the N-body simulation, but when having to process a large number of bodies, most of them are limited by the huge computational requirements. Our developed approach has the advantage and brings the novelty of harnessing the dynamic parallelism feature and the huge computational potential of the Pascal Compute Unified Device Architecture, offering in a reduced execution time the accurate states of the N interacting bodies: positions, velocities and accelerations. The proposed implementation of the N-body simulation, based on the Verlet Leapfrog Algorithm, proves to be a useful tool in numerous scientific fields, considering the high computational throughput, the obtained level of performance and efficiency.

REFERENCES

[1] L. Nyland, M. Harris, J. Prins, "Fast N-Body Simulation with CUDA", in GPU Gems 3, chapter 31, Addison Wesley, Boston, 2007.

[2] E. Elsen, M. Houston, V. Vishal, E. Darve, P. Hanrahan, V. Pande, "N-Body simulation on GPUs", in Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC '06), ACM, New York, 2006.

[3] J. Jeffers, J. Reinders, A. Sodani, "N-Body simulation", in Intel Xeon Phi Processor High Performance Programming, Morgan Kaufmann, Boston, 2016.

[4] J.A. Franco R., The N-Body Problem: Classic and Relativistic Solution: Corrections to: Newton's Gravitational Force for N>2, and Einstein's relativistic Mass & Energy, under a 3-D Vectorial Relativity approach, CreateSpace Independent Publishing Platform, North Charleston, 2016.

[5] T.Levi-Civita, The n-Body Problem in General Relativity, D. Reidel Pub. Co., Dordrecht, 1964.

[6] T. Burgess, The n-Body Problem, ChiZine Publications, Toronto 2013.

[7] K. Meyer, G. Hall, D. Offin, Introduction to Hamiltonian Dynamical Systems and the N-Body Problem (Applied Mathematical Sciences), Springer, New York, 2009.

[8] S. J. Aarseth, Gravitational N-Body Simulations: Tools and Algorithms (Cambridge Monographs on Mathematical Physics), Cambridge University Press, 2009.

[9] P.J. Teuben, "The Stellar Dynamics Toolbox NEMO", in: Astronomical Data Analysis Software and Systems IV, PASP Conf. Series, vol. 77, 1995.